Wireless Ad-Hoc Network Temperature Logging

Joe Schneider, Mike Schmitz, Rio Mascarenhas, Rajendra Katti

Wireless sensor networks of the past have had a few problems. The range of the radio transmitters has placed a limitation on how close one node must be to another one. The radio transmission power must be balanced so as to have enough power to reach the next node while also using an acceptable amount of battery power to do so. If a wireless network was in a star formation with each node connecting to a host, all nodes have to be in range of the host to communicate with the network at all. Two nodes might be right next to each other but unable to communicate because one cannot connect to the host. This issue needed to be addressed, and the obvious answer from above is to implement a peer-peer network, in which each node can effectively communicate with every other node within range.

However, implementing a peer-peer network is not as easy as it may look. Imagine a string or line of sensors all within the range of just two other sensors in the network. The node on one end of the string cannot directly communicate with the node on the other end of the line, but it can talk to its neighbor, who can talk to its neighbor, and eventually deliver the message to its intended destination. A network of this sort is called a hopping network because messages can be communicated to nodes in the network out of their range by hopping on those nodes that are close by. This is an easy idea, but complex to implement. Routing tables are necessary for nodes to know if there is a route to another node, and even then the node must know if that route includes itself and to whom it should transmit to continue the sending of the message. Now, imagine that the string of sensors has changed to a blob of sensors, all of which are mobile and may be in the range of many sensors. The problem suddenly just got a lot harder. Add to this the ability for the network to form itself without any outside information (called an "ad hoc" network), as well as the ability to add additional nodes at any time and have them completely assimilated into the network, and the problem suddenly explodes.

For our senior design project, we were asked to create an ad-hoc network of sensors. For our proof-of-concept design, we chose to measure the ambient temperature of the sensor; however the design to be given is easily adaptable to other sensing needs.

PROTOCOL STRUCTURE

A large portion of the development of an ad-hoc network goes into the protocol. Many ad-hoc protocols have been developed and are available to be implemented in the form of research publications. However, after doing research, many of these protocols were found to be unnecessarily complicated for our application. What was needed was a small protocol that did just what we needed it to do, and nothing more. In response, we developed our own, loosely based on the Direct Source Routing (DSR) protocol from Carnegie Mellon's Monarch project [1]. The protocol incorporates a simple packet structure, with only two types of packets necessary for full operation.

The protocol structure is given in Figure 1. Each portion of the protocol is described below:

Preamble -

The preamble is necessary for transmitter/receiver USART synchronization and receiver initialization. The first part of the preamble must consist of no less than 10 high bits, without any start or stop bits accustomed to USART communication. This causes the receiver's USART to reset and therefore perceive the next high to low transition as a start byte. The rest of the preamble consists of 3 bytes with the binary value "01010101." These three bytes are sent with the normal start and stop bytes and are necessary to charge the capacitor in the data slicer of the receiver to a specific level for optimum noise rejection during packet transmission. The three identical and consecutive bytes are also used to key the receiving node on the beginning of a packet transmission.

Address Byte –

The address byte identifies the destination of the packet and the type of packet being transmitted.

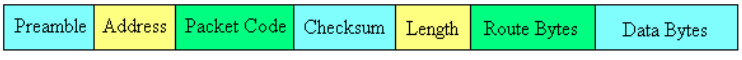


Figure 1: Protocol Structure

Packet Code Byte –
The packet code byte is necessary to prevent the flooding of the network with Route Discovery packets. The packet code prevents infinite loops in the network by including a random number in the packet to serve as a packet identifier.

Checksum Byte –
The checksum byte is used for error detection in the packet. The checksum equals the exclusive-or of all bytes in the packet excluding the preamble and the checksum byte itself.

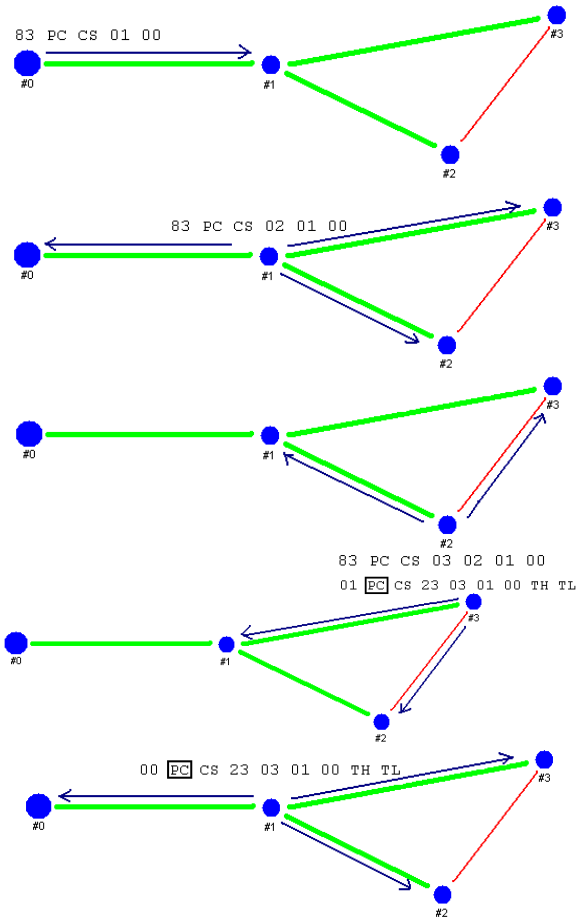
Length Byte –
The length byte is necessary for the receiver to identify the number of routing addresses and data bytes contained in the packet, and the total length of the packet.

Route Bytes –
The route bytes comprise the packet's route cache.

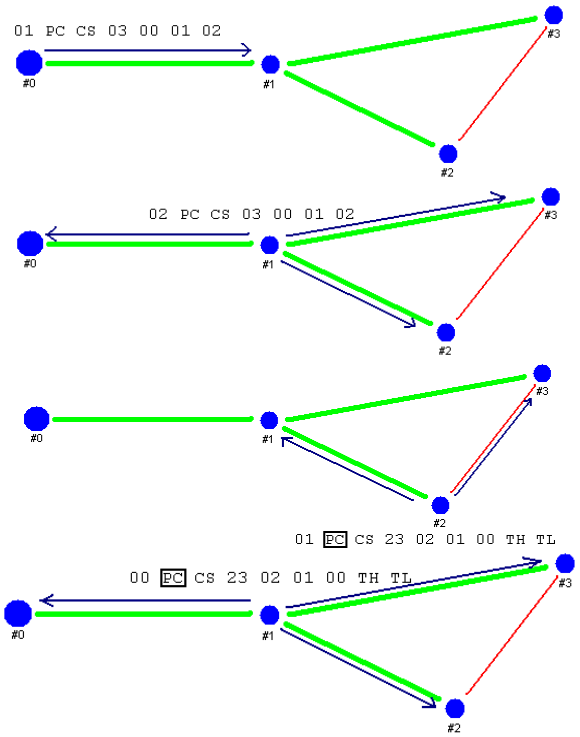
Data Bytes –
The data bytes contain the information being transferred by the network.

ROUTE DISCOVERY PACKETS
When the base unit needs to initiate communication with a given node, it first searches its memory for a suitable route. If no route is found or if the stored route no longer works due to an ever-changing network configuration, the base unit must generate and transmit a Route Discovery packet. The purpose of this packet is to travel throughout the entire network, collecting route data in the process, until it reaches the intended node. Upon reception, the desired node will analyze the collected route data and generate a Route Aware packet using the collected route data and, at the same time, append any necessary data. This packet will propagate through the network, passing only through the nodes specified in the route data, until it reaches the base unit. The base unit can then analyze the received Route Aware packet and store the included route data in memory. The base unit now has a working route to the given node. Any included data in the packet is also available for the base unit's use.

Route information is collected by the Route Discovery by having every node receiving the packet modify it. When a node receives the Route Discovery, it first checks the address byte



Example Route Discovery from Node #0 to Node #3



Example Route Aware from Node #0 to Node #2

Figure 2: Protocol Examples

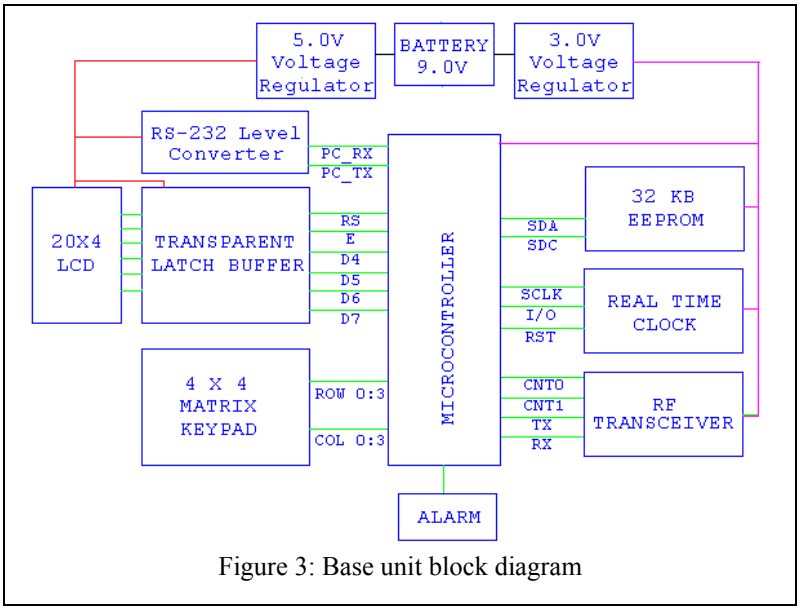


Figure 3: Base unit block diagram

to see if it is the intended destination of the packet. If it is not, the node must update the route bytes. It does this by inserting its own address before the first route byte in the packet and incrementing the length byte.

ROUTE AWARE PACKETS

Once the base unit acquires and stores to memory a working route to a node through the use of a Route Discovery, it can communicate to the node using the other type of packet called Route Aware. The benefit to using a Route Aware is that the packet already contains the route information necessary to directly deliver the packet to the intended node, whereas a Route Discovery would be transmitted throughout the entire network. Therefore, a Route Aware requires fewer transmissions thereby conserving power and reducing unnecessary noise in the network. Likewise, a node can generate a Route Aware packet destined for the base unit. However, since nodes do not store route information in memory, a route back to the base unit must be collected from elsewhere. The easy solution to this is to use the route information contained in the Route Discovery or Route Aware packet received by the node from the base unit.

PROTOCOL EXAMPLES

To help illustrate the protocol, the series of pictures in Figure 2 have been drawn up. In each figure, PC represents a packet code, CS the checksum, and TH and TL two bytes of temperature data. Two examples of network operations are shown, one for a route discovery, and the other for a route aware packet. Packet contents at each point in time are shown in text next to the transmitting node. Green lines represent primary data paths, while red lines represent secondary data paths.

CIRCUIT DESCRIPTION

After hashing out the details of the protocol, we began designing and building the hardware to implement our protocol. Figures 3 and 4 show the block diagrams of our base unit and nodes respectively.

Base Unit -

The base unit is powered by a 9V battery or any 9V DC source which is regulated to 5V for some of the circuit, and 3V for other parts. The reason for this is while the RF transceiver used can only be run off of 3V, the LCD and RS-232 chip require a 5V source. Special care had to be taken when designing the circuitry from the microcontroller, running on 3V, to the 5V components, and hence, the transparent latch buffer. Not shown on the block diagram is additional circuitry to switch levels on the RS-232 converter.

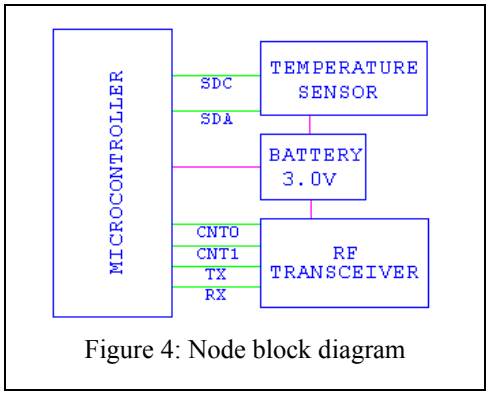


Figure 4: Node block diagram

The microcontroller controls input via a 4x4 matrix keypad, outputs information to the user using both the LCD (standard 20x4 character LCD) and an RS-232 connection (MAX232), communicates with the network using the RF transceiver (RFM DR3000), time stamps and logs data using the RTC (DS1302) and the EEPROM (24LC256), and sounds an alarm if temperatures fall outside of a customizable range.

Node -
The nodes are similar to the base unit in their use of the same transceiver (RFM DR3000) to communicate with the network and a microcontroller to process the network protocol. Additionally, the nodes each have a temperature sensing IC (DS1631) to measure the ambient temperature of the node. All components in the nodes are powered by a 2xAA battery 3V source.

USER INTERFACE
The user interface on the base unit consists of a LCD and keypad. The user operates the base unit by navigating through the menus on the LCD. The two screens of the main menu are shown below, and each option is described in detail.

```
Node temp
Start collection
View data
Connect
```

```
5. Set clock
6. Temp alarm range
7. Delete all memory
```

Node temp –
This option performs a single temperature collection from one node in the network. The user is prompted to enter the desired node's address, a temperature request will be made for the node, and the corresponding temperature will be displayed on the LCD. If communication fails with the node, "Communication Failure" will be displayed on the LCD. The collected temperature will not be stored in memory and is not available for download to a computer.

Start collection –
This option performs periodic temperature collection from a group of nodes. All collected temperatures along with the times of collection will be stored in memory and are available for download to a computer. If any temperature is outside of the temperature alarm range, the alarm will be activated. When this option is selected, the user will be prompted to enter the addresses of all nodes that should be polled, followed by the polling frequency. The polling frequency can range from continuous to once every 256 minutes with a resolution of one second. Once temperature collection begins, the LCD will display only the most recent collected temperature, the address of the node it came from, and the time of collection. However, if a collected temperature falls outside the temperature alarm range, only this temperature will be displayed on the LCD. It will remain on the LCD until replaced by a new temperature that falls outside the temperature alarm range.

View data –
This option allows the user to view on the LCD the temperatures stored in memory during periodic collection. The user will be prompted to enter the address of the desired node, and then the stored temperatures and collection times for that node will be displayed on the LCD in the order of collection beginning with the most recent.

Connect –
This option allows a computer to control the base unit. The computer is able to download the temperatures and times stored in memory, download the routes to nodes in the network, read and set the base unit's clock, and perform

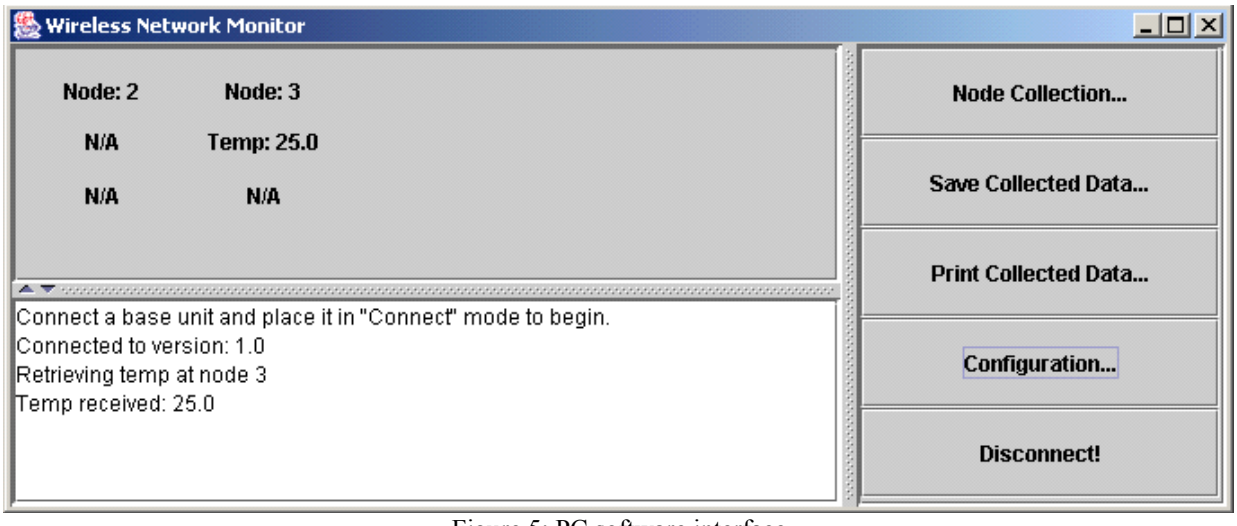


Figure 5: PC software interface

single temperature collections from individual nodes.

Set clock –

This option allows the user to read and set the base unit's time and date.

Temp alarm range –

This option allows the user to read and set the minimum and maximum acceptable temperatures. If any collected temperature falls outside this range, the alarm will be activated.

SOFTWARE CONNECTION

Figure 5 shows a screen capture from the PC software developed for the project. The software was created in Java because of its price and interoperability. The PC software can perform any action the base unit can, and can perform some additional tasks such as plotting the topology of the network and logging data indefinitely to a text file.

FINAL PRODUCT

Photos 1 and 2 show the final base unit and node hardware. The completed project functioned as a wireless ad-hoc network as expected. All code was written in C using CCS's (Custom Computer Services) PCM compiler, allowing for easy modifications in the future. This project could easily be adapted to a host of sensing applications.

Joe Schneider is an undergraduate student at North Dakota State University. You may reach him at Joseph.Schneider@ndsu.nodak.edu.

Mike Schmitz is an undergraduate student at North Dakota State University. You may reach him at Michael.Schmitz@ndsu.nodak.edu.

Rio Mascarenhas is an undergraduate student at North Dakota State University. You may reach him at Rio.Mascarenhas@ndsu.nodak.edu.

Dr. Rajendra Katti earned his Ph.D. in Electrical Engineering from Washington State University. Currently, he is an associate professor at North Dakota State University. You may reach him at Rajendra.Katti@ndsu.nodak.edu.

REFERENCES

[1] *The CMU Monarch Project*. 28 Feb 2002. Available at http://www.monarch.cs.cmu.edu.

Photo 1: Base unit

Photo 2: Nodes